04

Para saber más: ¿DTO o entidades?

Usamos DTO para representar los datos que recibimos y devolvemos a través de la API, pero probablemente se esté preguntando "¿Por qué, en lugar de crear un DTO, no devolvemos directamente la entidad JPA en el Controller?". Para hacer esto, simplemente cambie el método `list` en el Controller a:

```
@GetMapping
public List<Medico> listar() {
    return repository.findAll();
}
```

COPIA EL CÓDIGO

De esa forma, el código sería más ligero y no necesitaríamos crear el DTO en el proyecto.

Pero, ¿es esto realmente una buena idea?

Problemas de recepción/devolución de la entidad JPA

De hecho, es mucho más simple y cómodo no usar DTO, sino tratar directamente con entidades JPA en los Controllers. Sin embargo, este enfoque tiene algunas desventajas, incluida la vulnerabilidad de la aplicación a los ataques de **Mass Assignment**.

Uno de los problemas es el hecho de que, al devolver una entidad JPA en un método del Controller, Spring generará el JSON que contiene **todos** sus atributos, y este no siempre es el comportamiento que queremos.

Eventualmente podemos tener atributos que no queremos que sean devueltos en el JSON, ya sea por razones de seguridad, en el caso de datos *sensibles*, o incluso porque no son utilizados por clientes API.

Uso de la anotación `@JsonIgnore`

En esta situación, podríamos usar la anotación `@JsonIgnore`, que nos ayuda a ignorar ciertas propiedades de una clase Java cuando se serializa en un objeto JSON.

Su uso consiste en agregar la anotación a los atributos que queremos ignorar cuando se genera el JSON. Por ejemplo, supongamos que tenemos una entidad JPA 'Empleado', en la que queremos ignorar el atributo 'salario':

```
@Getter
@NoArgsConstructor
@EqualsAndHashCode(of = "id")
@Entity(name = "Empleado")
@Table(name = "empleados")
public class Empleado {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String email;

    @JsonIgnore
    private BigDecimal salario;
```

```
    //restante del código omitido…
}
```

COPIA EL CÓDIGO

En el ejemplo anterior, el atributo 'salario' de la clase 'Empleado' no se mostrará en las respuestas JSON y el problema estaría resuelto.

Sin embargo, puede haber algún otro endpoint de la API en el que necesitemos enviar el salario de los empleados en el JSON, en cuyo caso tendríamos problemas, ya que con la anotación `@JsonIgnore` tal atributo **nunca** se enviará en el JSON, y al eliminar la anotación se enviará el atributo **siempre**. Por lo tanto, perdemos la flexibilidad de controlar cuándo se deben enviar ciertos atributos en el JSON y cuándo no.

DTO

El patrón DTO (*Data Transfer Object*) es un patrón arquitectónico que se usó ampliamente en aplicaciones Java distribuidas (arquitectura cliente/servidor) para representar los datos que eran enviados y recibidos entre aplicaciones cliente y servidor.

El patrón DTO puede (y debe) usarse cuando no queremos exponer todos los atributos de alguna entidad en nuestro proyecto, una situación similar a los salarios de los empleados que discutimos anteriormente. Además, con la flexibilidad y la opción de filtrar qué datos se transmiten, podemos ahorrar tiempo de procesamiento.

Bucle infinito que causa `StackOverflowError`

Otro problema muy recurrente cuando se trabaja directamente con entidades JPA ocurre cuando una entidad tiene alguna auto-relación o relación bidireccional. Por ejemplo, considere las siguientes entidades JPA:

```
@Getter
@NoArgsConstructor
@EqualsAndHashCode(of = "id")
@Entity(name = "Producto")
@Table(name = "productos")
public class Producto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String descripcion;
    private BigDecimal precio;

    @ManyToOne
    @JoinColumn(name = “id_categoria”)
    private Categoria categoria;

    //restante del código omitido…
}
```

COPIA EL CÓDIGO

```
@Getter
@NoArgsConstructor
@EqualsAndHashCode(of = "id")
@Entity(name = "Categoria")
@Table(name = "categorias")
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    private String nombre;

    @OneToMany(mappedBy = "categoria")
    private List<Producto> productos = new ArrayList<>();

    //restante del código omitido…
}
```

COPIA EL CÓDIGO

Al devolver un objeto de tipo 'Producto' en el Controller, Spring tendría problemas para generar el JSON de este objeto, lo que provocaría una excepción de tipo 'StackOverflowError'. Este problema ocurre porque el objeto producto tiene un atributo de tipo `Categoría`, que a su vez tiene un atributo de tipo `Lista<Producto>`, lo que provoca un bucle infinito en el proceso de serialización a JSON.

Este problema se puede resolver usando la anotación `@JsonIgnore` o usando las anotaciones `@JsonBackReference` y `@JsonManagedReference`, pero también se puede evitar usando un DTO que represente solo los datos que se deben devolver en el JSON.